Fault Tolerance for Heterogeneous Systems

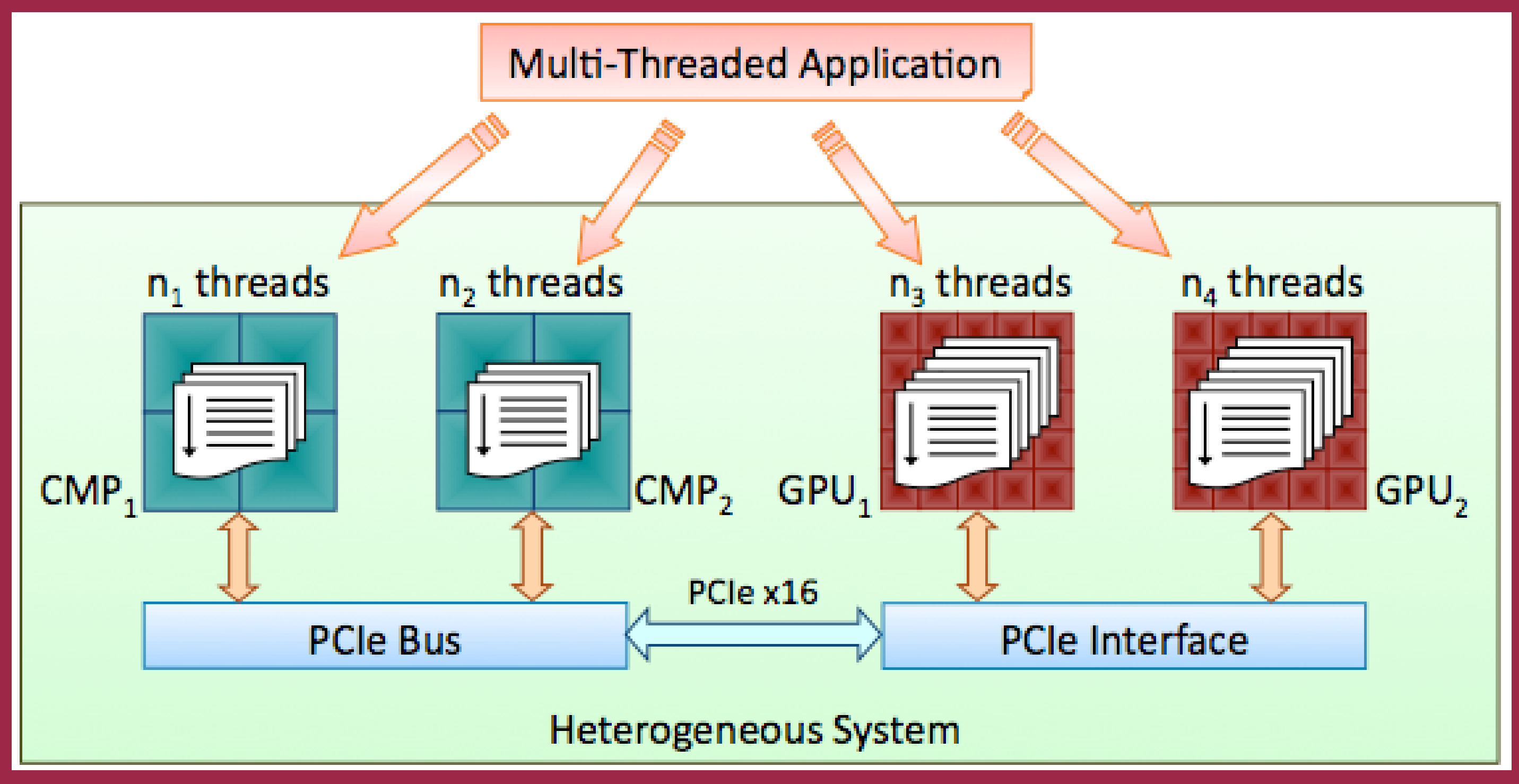
THE Ames Laboratory
Creating Materials & Energy Solutions
U.S. DEPARTMENT OF ENERGY
SCL

Motivation: Heterogeneous systems (HSs) composed of Chip Multicore Processors (CMPs) and Graphics Processing Units (GPUs) have proven they can achieve interesting speedups on scientific applications. Like any computer engine of today, GPUs are vulnerable to transient and permanent failures. Current checkpoint/restart techniques are not suitable for HSs with GPUs, primarily due to the natural differences imposed by the hardware design, the memory subsystem architecture, the massive number of threads, and the limited amount of synchronization among threads. Therefore, a checkpoint/restart technique suitable for heterogeneous systems is needed.

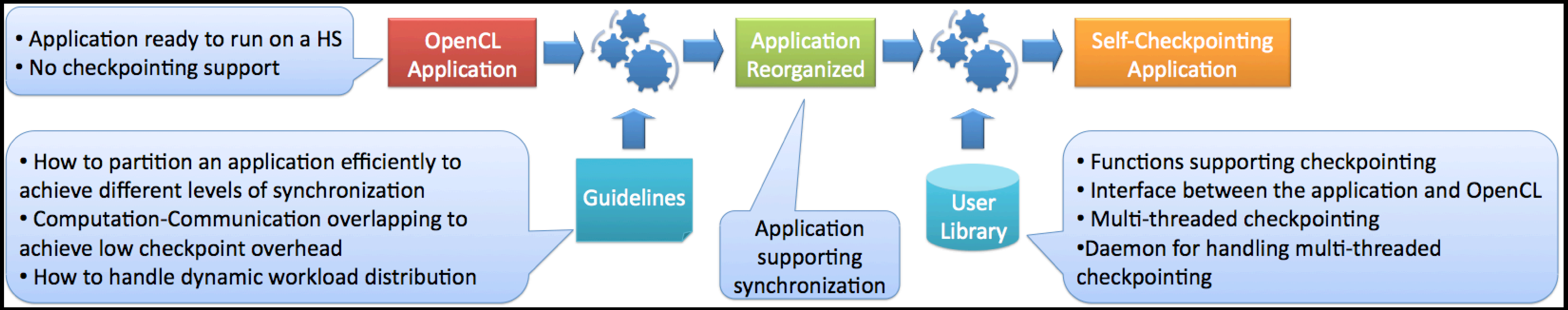
Goal: To design, implement, and analyze a complete framework for application-level checkpointing on heterogeneous systems.

Challenges: The proposed framework should:

- Support applications running on all CMPs and GPUs present on the HS
- Not require global synchronization – too costly to synchronize thousands of in-flight threads
- Be flexible to support computational devices discovery and workload distribution at runtime
- Support different programming models: SIMT for GPUs and MIMD for CMPs
- Impose low checkpoint overhead
- Avoid bus bottlenecks at the buses - slow device to host data transfer



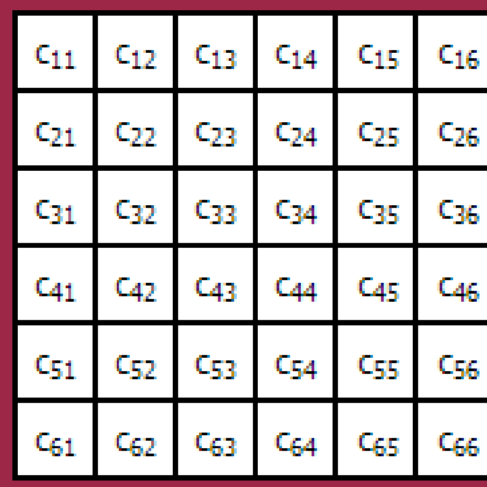
The Proposed Framework: Application-Level Checkpointing for Heterogeneous Systems



• Application ready to run on a HS
• No checkpointing support
OpenCL Application
Application Reorganized
Self-Checkpointing Application
• How to partition an application efficiently to achieve different levels of synchronization
• Computation-Communication overlapping to achieve low checkpoint overhead
• How to handle dynamic workload distribution
Guidelines
Application supporting synchronization
User Library
• Functions supporting checkpointing
• Interface between the application and OpenCL
• Multi-threaded checkpointing
•Daemon for handling multi-threaded checkpointing

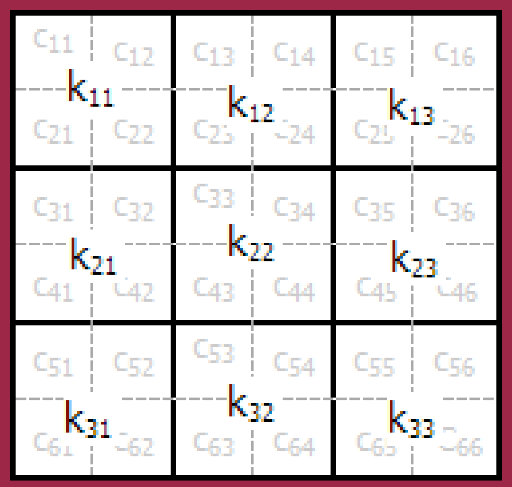
Preliminary Results: Application partition, Computation-Communication overlapping and Checkpoint Overhead was analyzed on two CUDA applications: Dense Matrix Multiplication and a Grid-Type application.

Dense Matrix Multiplication



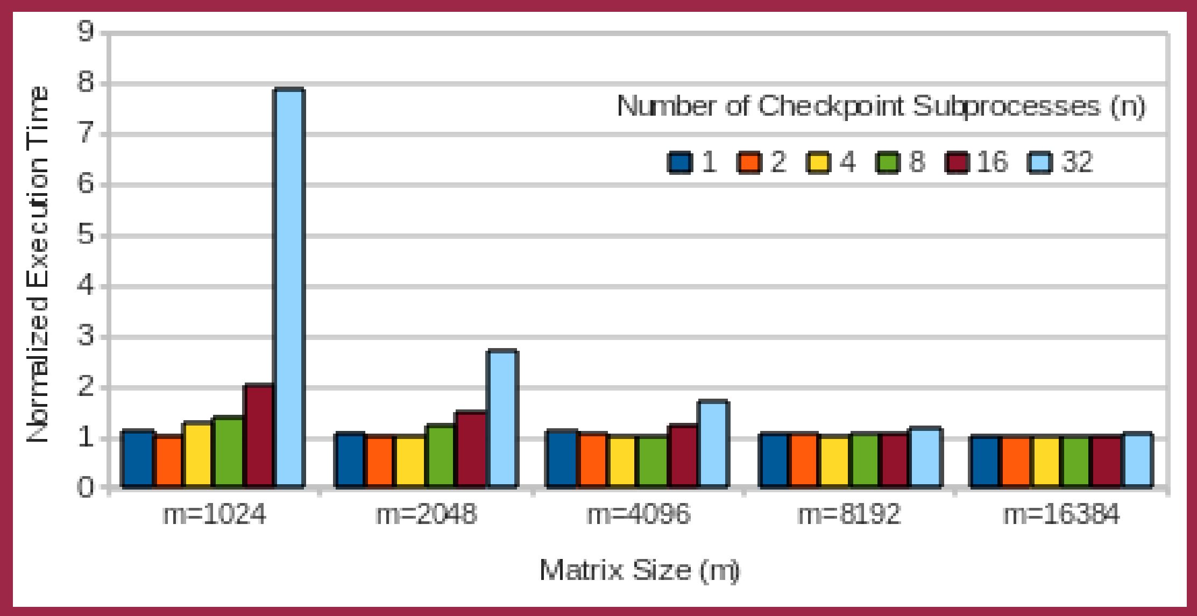
C_{11} C_{12} C_{13} C_{14} C_{15} C_{16}
C_{21} C_{22} C_{23} C_{24} C_{25} C_{26}
C_{31} C_{32} C_{33} C_{34} C_{35} C_{36}
C_{41} C_{42} C_{43} C_{44} C_{45} C_{46}
C_{51} C_{52} C_{53} C_{54} C_{55} C_{56}
C_{61} C_{62} C_{63} C_{64} C_{65} C_{66}

(a) A mxm matrix (m=6)



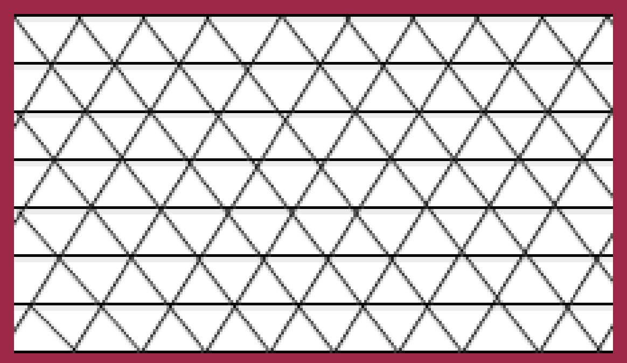
k_{11} k_{12} k_{13}
k_{21} k_{22} k_{23}
k_{31} k_{32} k_{33}

(b) Division into nxn submatices (n=3)

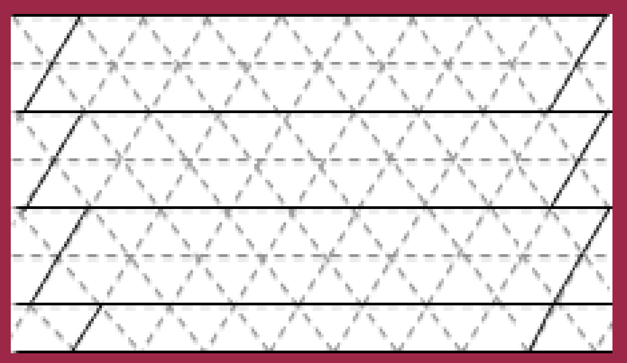


(c) Performance Overhead achieved by the Checkpointed version compared to the Non-Checkpointed version of the Dense Matrix Multiplication

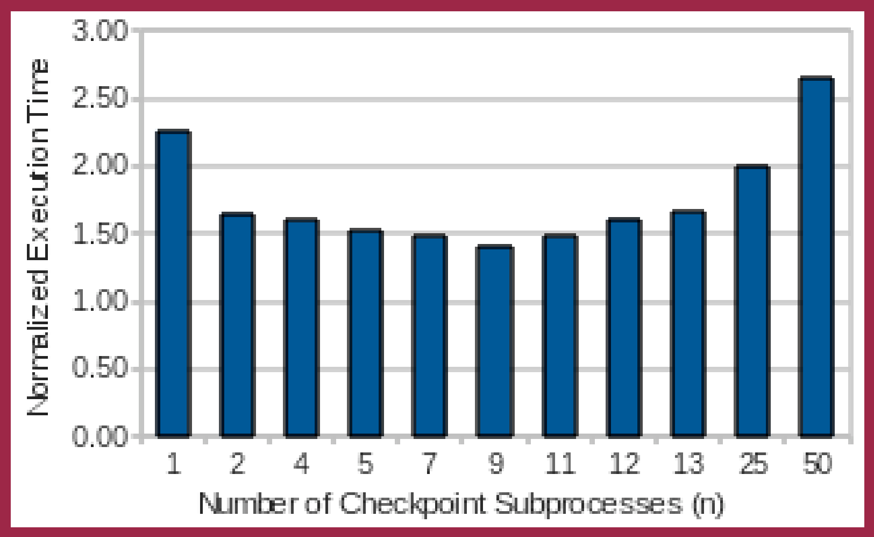
Grid-Type Application



(a) Grid representation of an application



(b) Division into smaller grids. Each smaller grid is a checkpoint subprocess



(c) Performance Overhead achieved by dividing the checkpoint process into smaller subprocesses

Conclusions:

- Overlapping GPU computation with checkpointing achieves low checkpoint overhead
- Partitioning the application achieves good performance until a point where there is not enough computation to keep the hardware busy, and therefore the execution time starts to increase

Contact Information:
Lizandro Solano-Quinde, Brett Bode
{ lsolano, brett }@scl.ameslab.gov